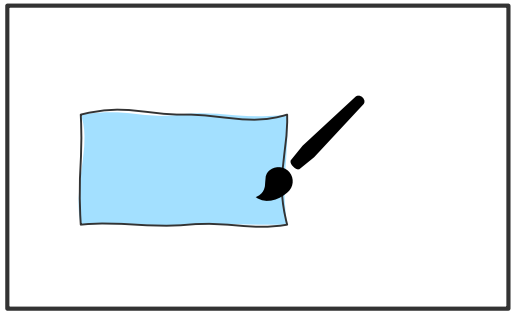
Ambiguous, Informal, and Unsound:
Metaprogramming for Naturalness

Toni Mattis, Patrick Rein, Robert Hirschfeld
Software Architecture Group
Hasso Plattner Institute, University of Potsdam, Germany

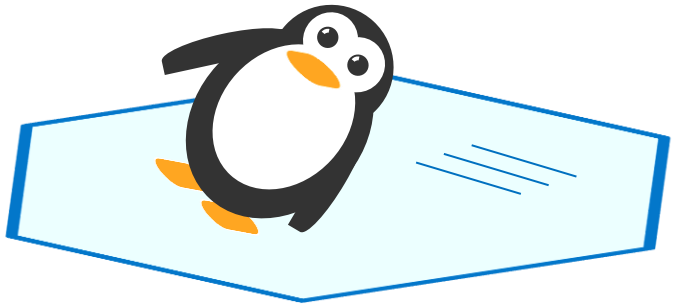
META'19 20 Oct. 2019, Athens, Greece

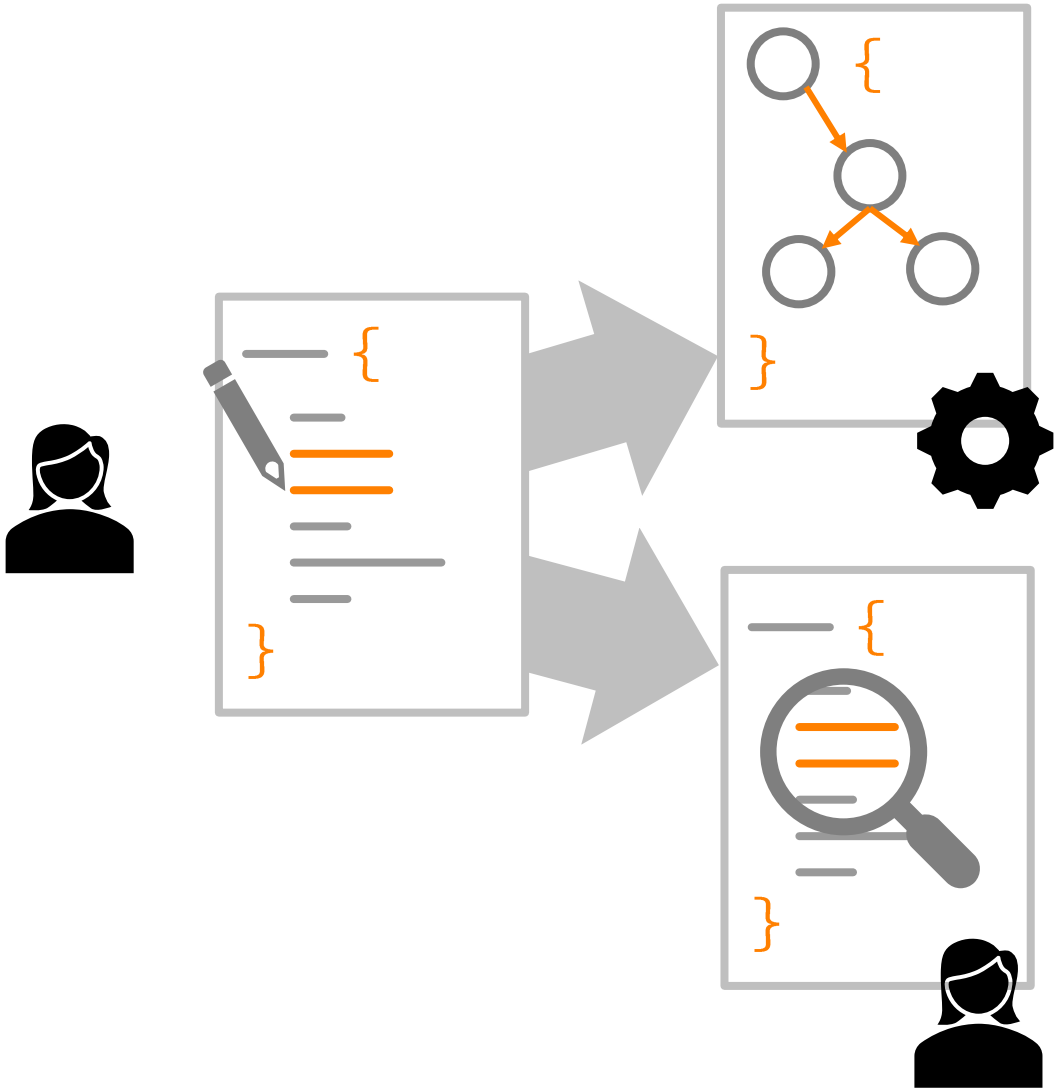
```
Rectangle >> drawOn: aCanvas
  self visible ifTrue:
   [aCanvas paint: self bounds
            color: self color]
```



```
Rectangle >> drawOn: aCanvas
  self visible ifTrue: [aCanvas paint: self bounds
       color:  self color                                    ]
```

```
Penguin >> slipOn: anIceFloe
  self clumsy ifTrue: [
    anIceFloe paint: self bounds
              color: self color]
```



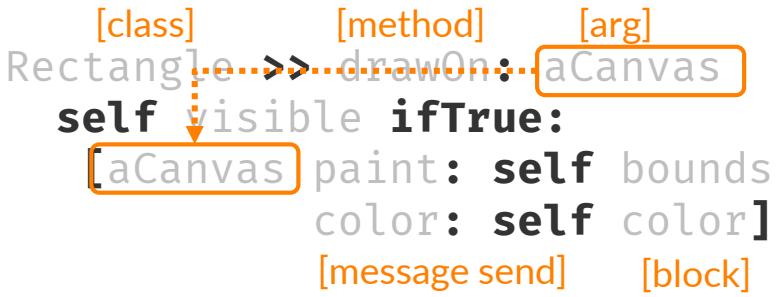


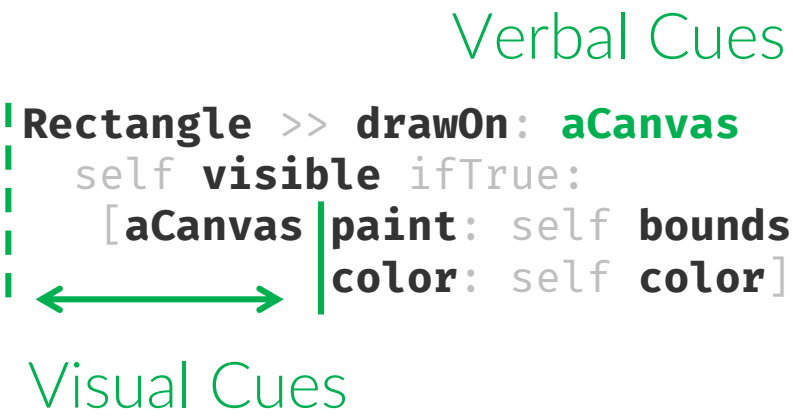
Systems
runtimes, tools

Humans
organization, future self

Primary notation
Behaviorally significant

Secondary notation
Perceptually significant





Primary notation
Behaviorally significant

- Formal, unambiguous
- Soundness guarantees

- Accessible through Metaprogramming

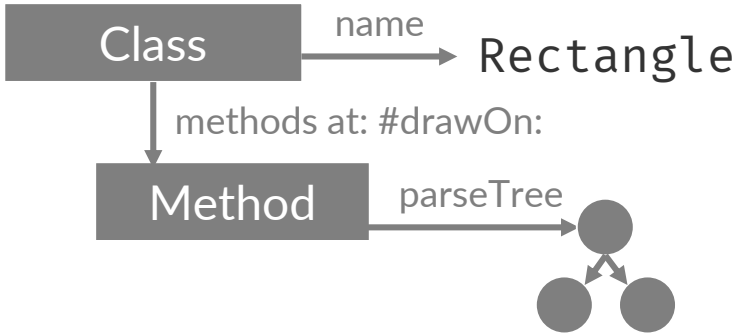
Secondary notation
Perceptually significant

- Informal, ambiguous
- Used inconsistently

- Subject to (**approximate**) interpretation

Primary notation
Behaviorally significant

```
Rectangle >> drawOn: aCanvas
  self visible ifTrue:
    [aCanvas paint: self bounds
             color: self color]
```



» Accessible through Metaprogramming

Secondary notation
Perceptually significant

```
Rectangle >> drawOn: aCanvas
  self visible ifTrue:
    [aCanvas paint: self bounds
             color: self color]
```

» Subject to (**approximate**) interpretation

Examples of Secondary Notation

Names

```
aCanvas
```

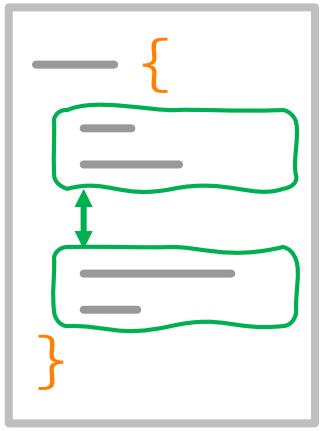
a canvas

Comments

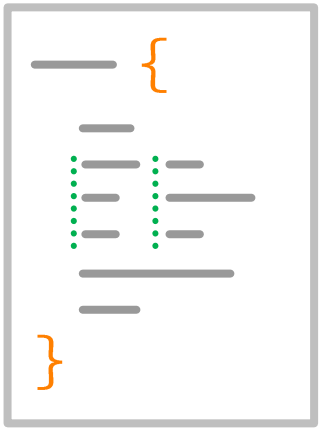
```
"Re-draw when
invalidated"
```

Values

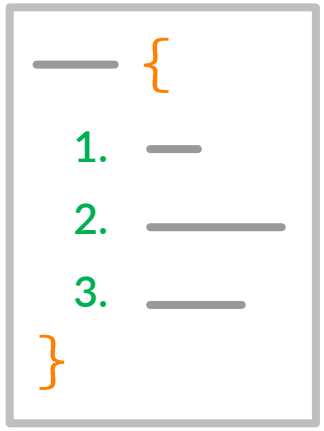
```
404

'error.log'
```



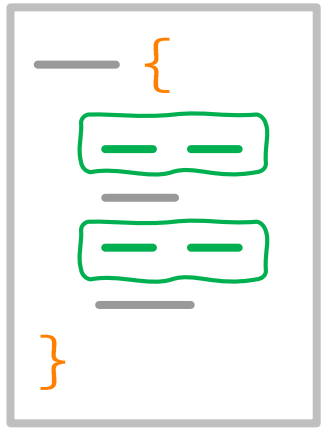
Space



Alignment

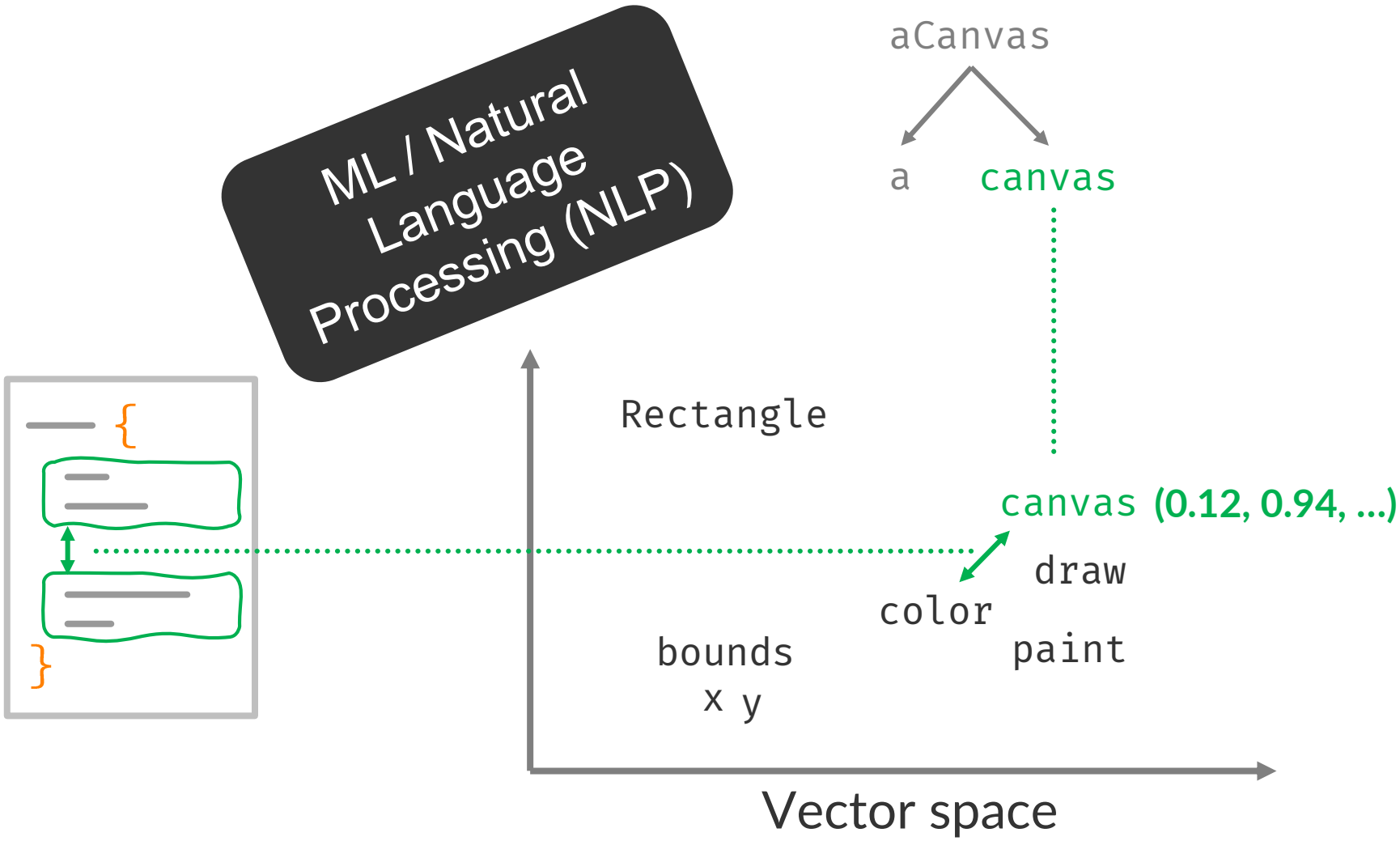


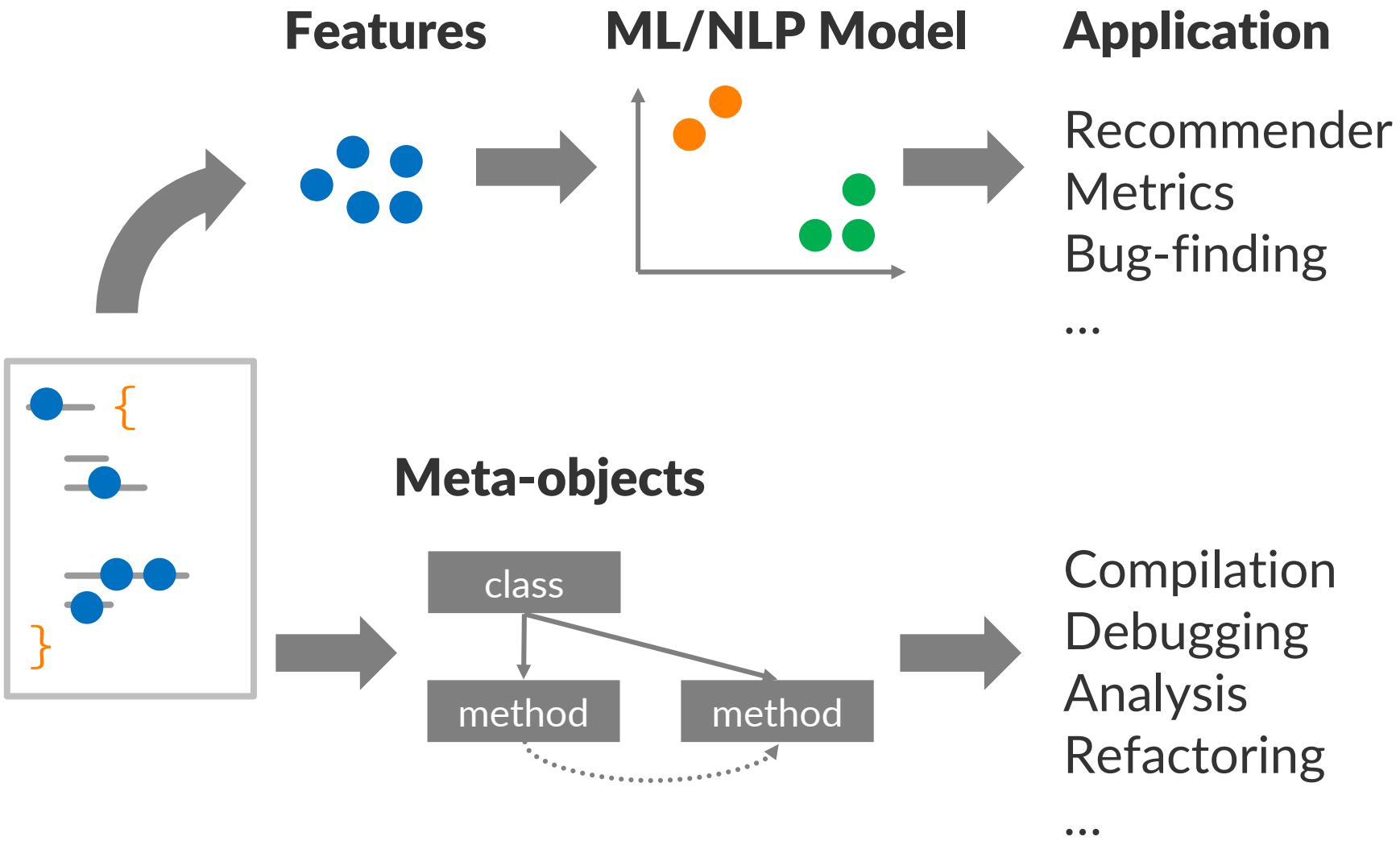
Order

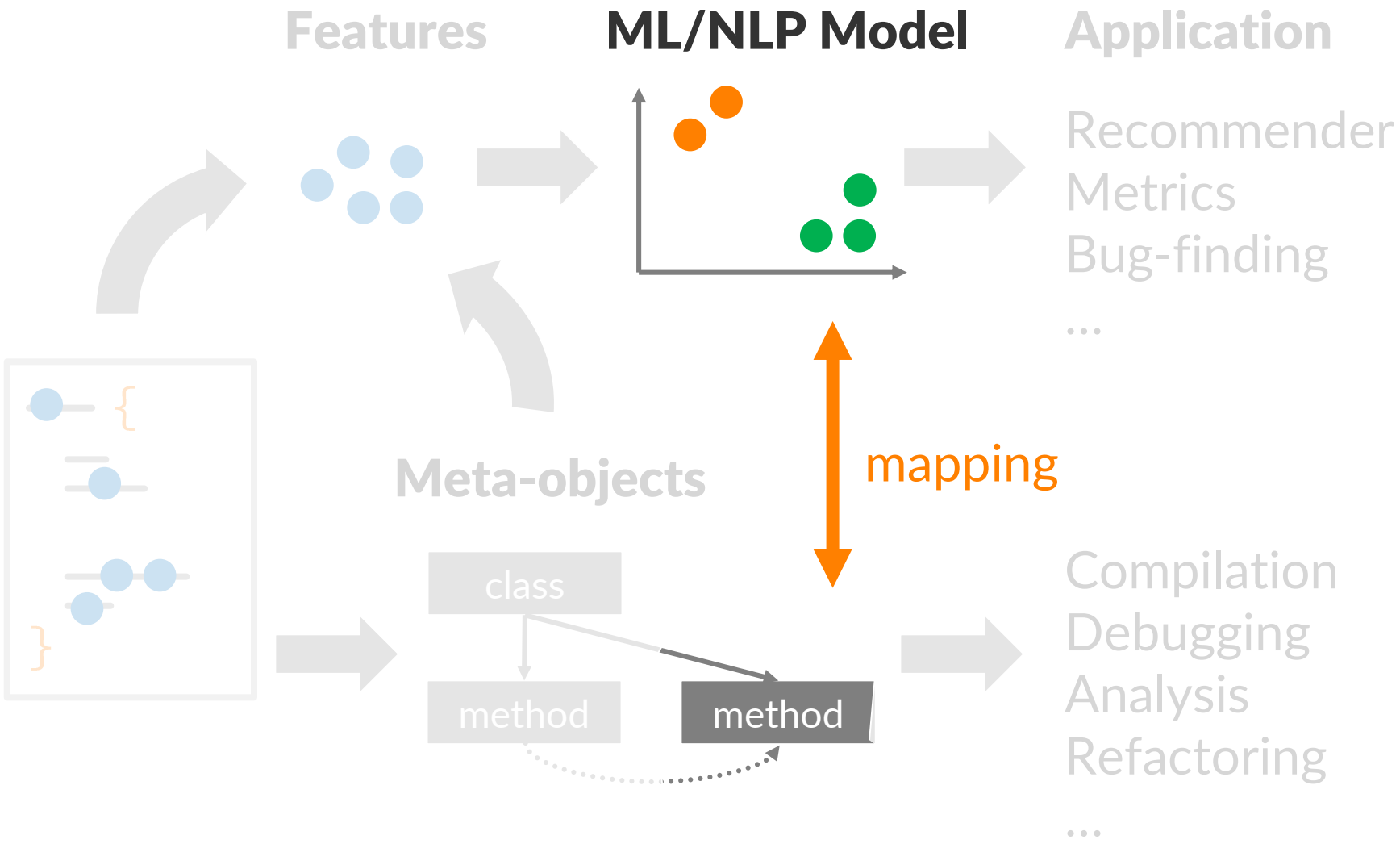


Co-occurrence

Machine Learning (ML) Example: Word Embedding



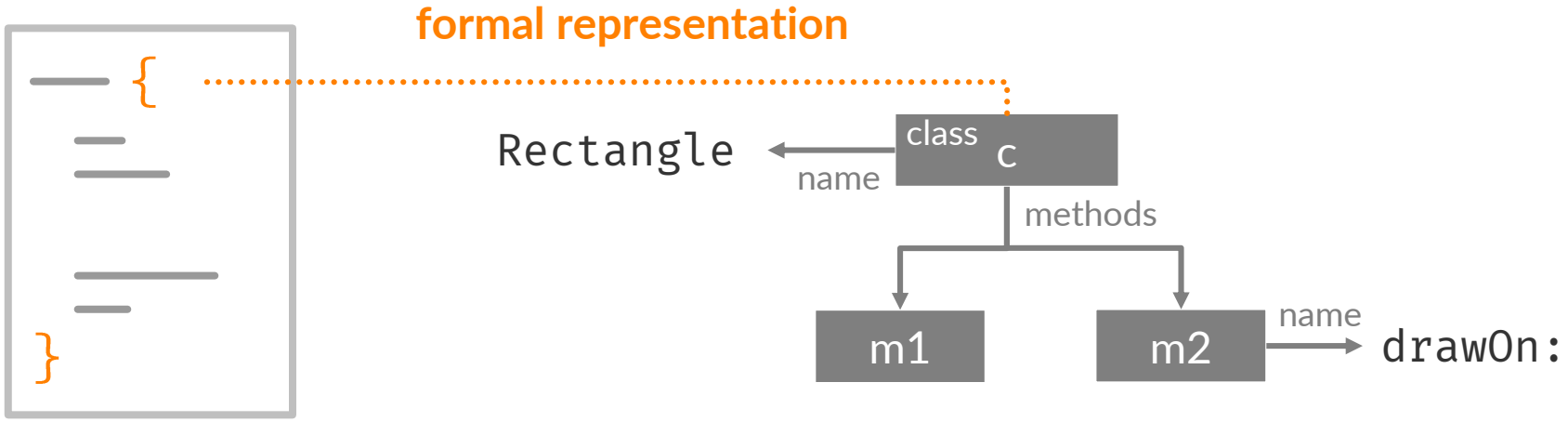

Features
ML/NLP Model
Application
Recommender
Metrics
Bug-finding
...
Meta-objects
class
method
method
Compilation
Debugging
Analysis
Refactoring
...


Features
ML/NLP Model
Application
Recommender
Metrics
Bug-finding
...
mapping
Meta-objects
class
method
method
Compilation
Debugging
Analysis
Refactoring
...

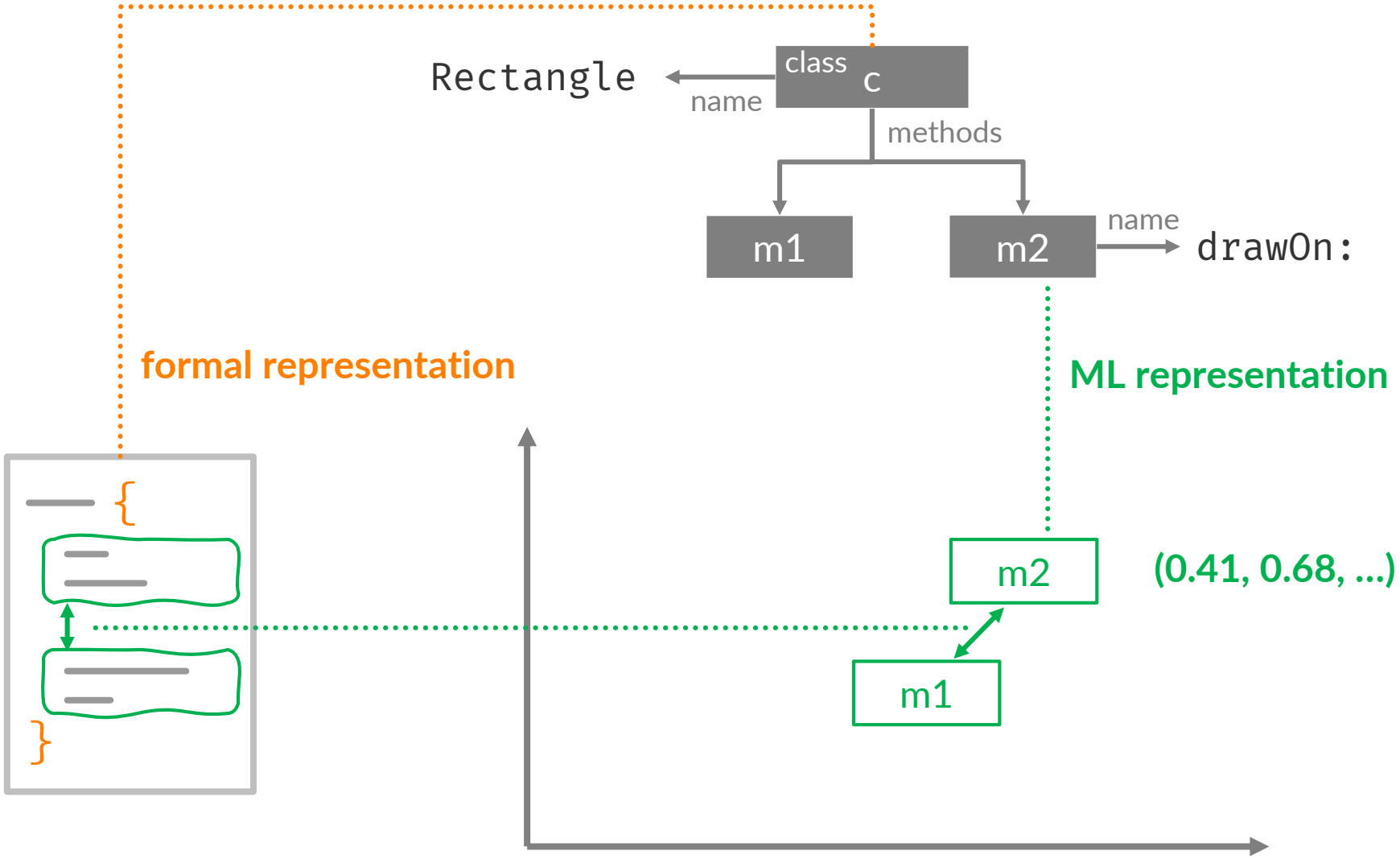
Meta-objects

Reified representations of a program's (formal) elements

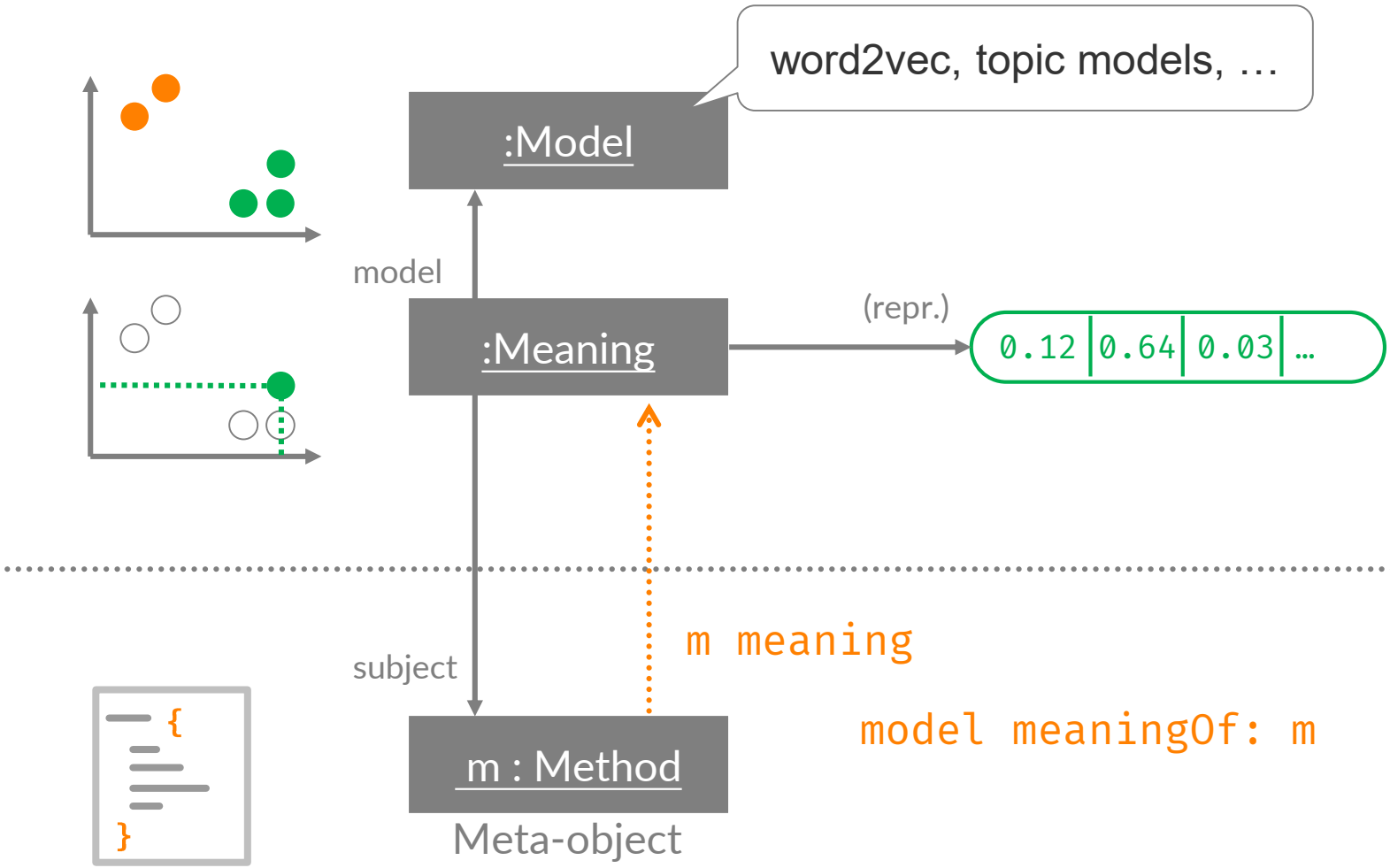
`Rectangle`	a Class
`m := Rectangle methodDict at: #drawOn:`	a CompiledMethod
`m parseTree body first`	The first statement of m



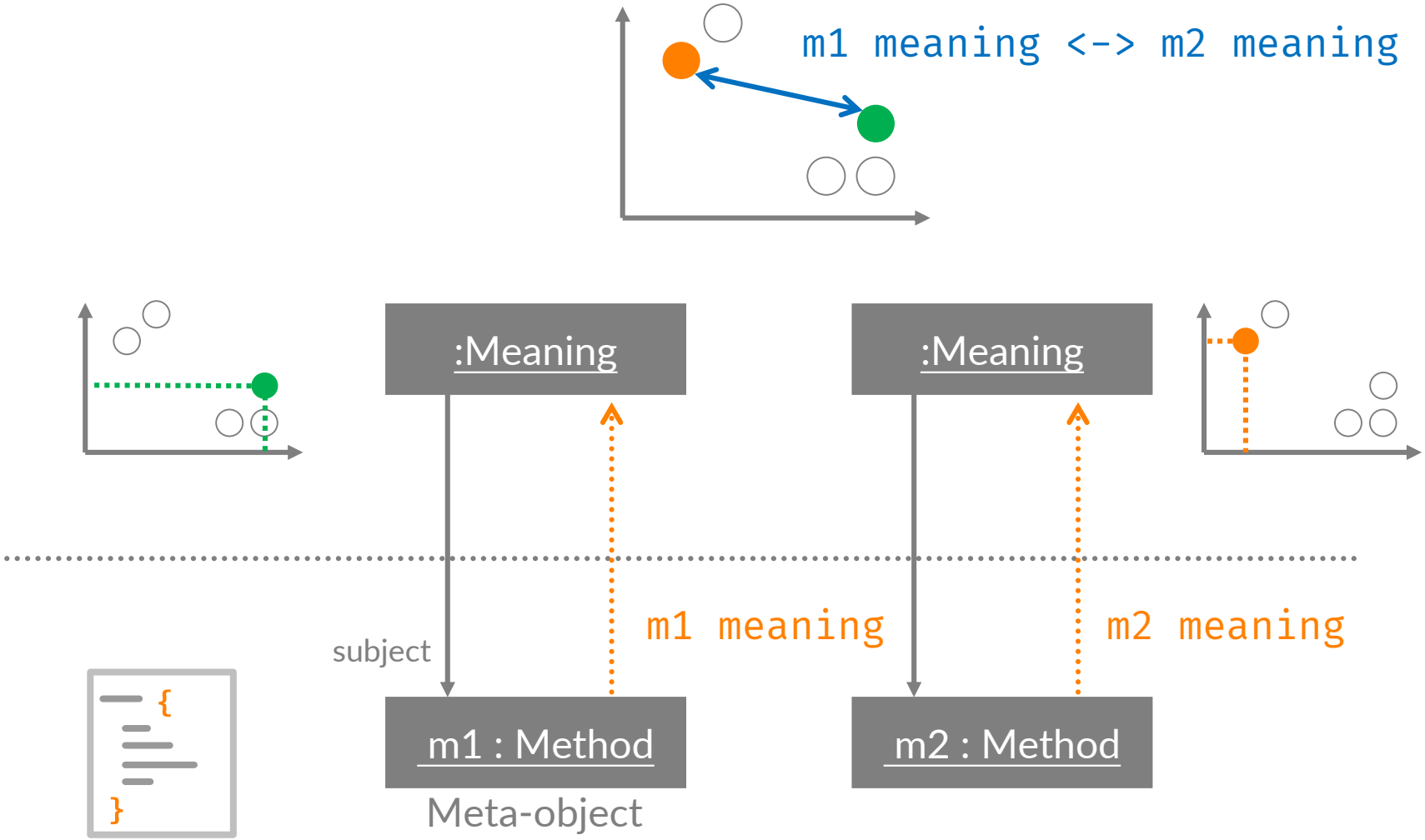
Representing Meta-objects



A Framework for “Meaning”

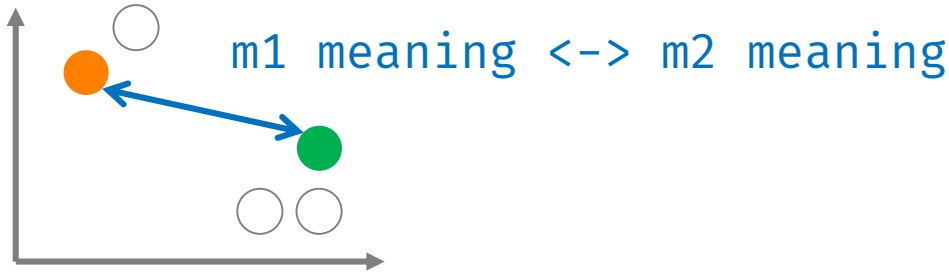


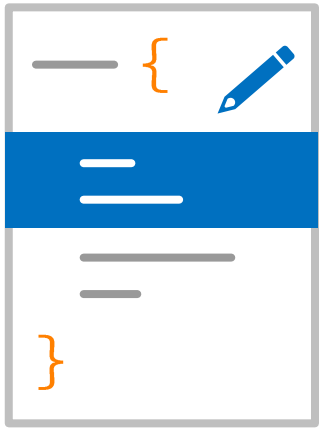
Comparing Meanings

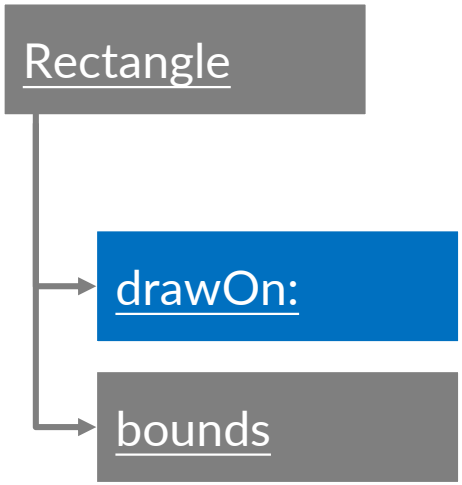


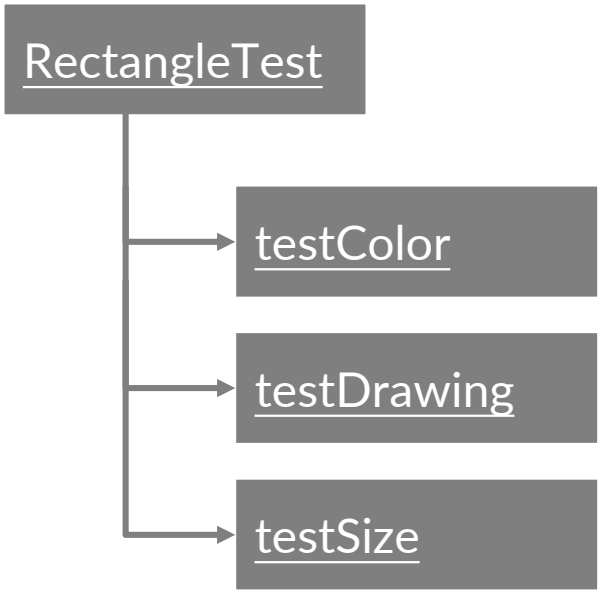
Comparing Meanings

Test Prioritization



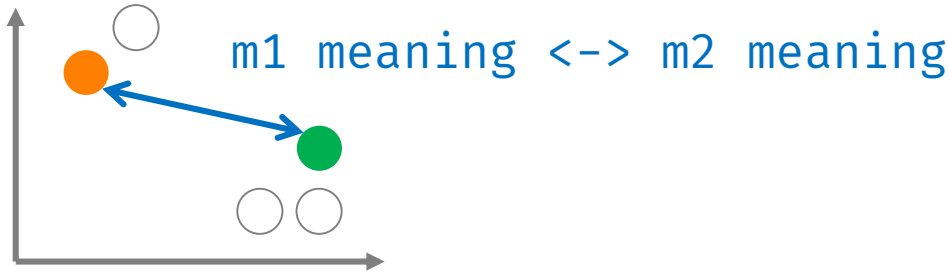


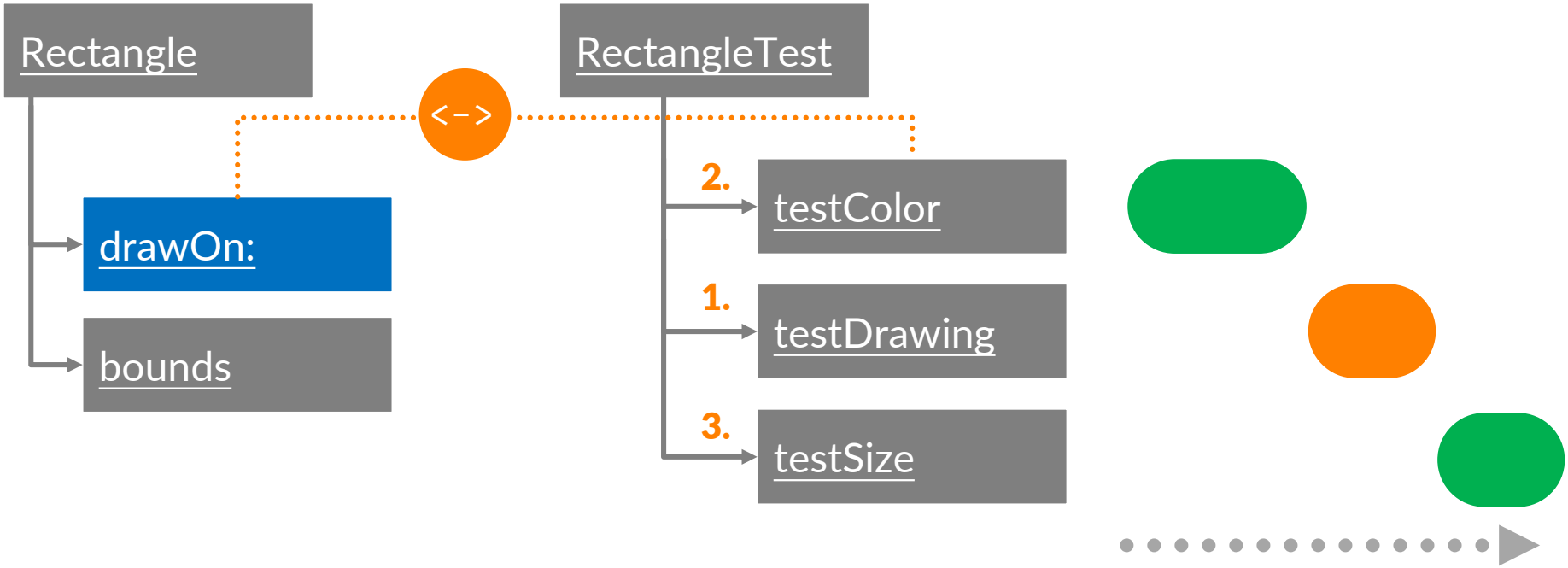




Comparing Meanings

Test Prioritization

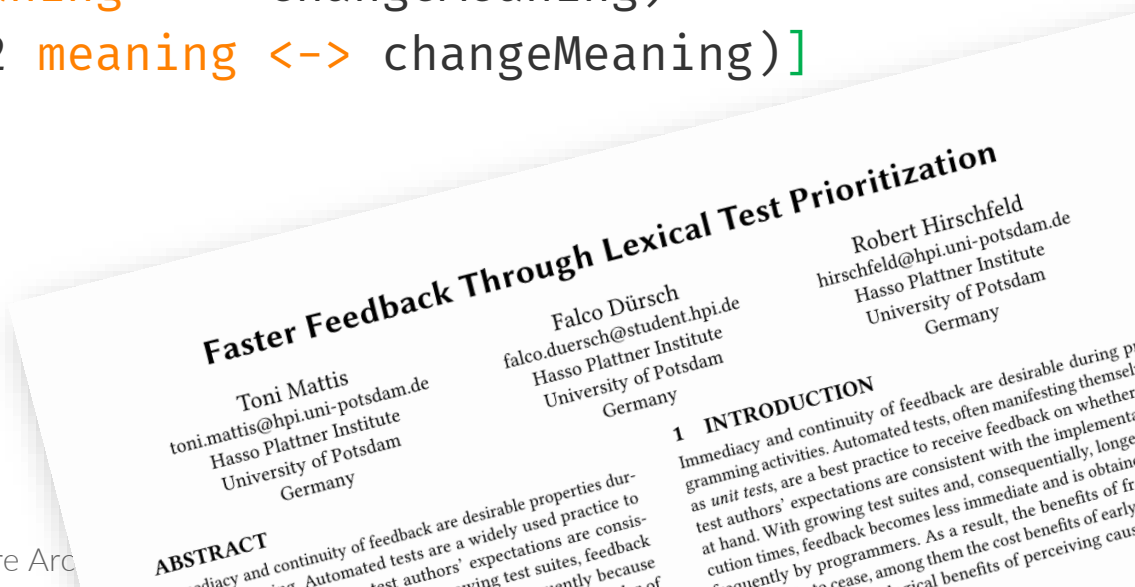




Implementation

Replace `self tests` in the `run` method with call to:

```
TestSuite >> prioritizedTestsGiven: aMethodChange
  | changeMeaning |
  changeMeaning := aMethodChange newMethod meaning.
  ^ self tests sorted: [:testMethod1 :testMethod2 |
      (testMethod1 meaning <-> changeMeaning)
        > (testMethod2 meaning <-> changeMeaning)]
```


Faster Feedback Through Lexical Test Prioritization

Toni Mattis
toni.mattis@hpi.uni-potsdam.de
Hasso Plattner Institute
University of Potsdam
Germany

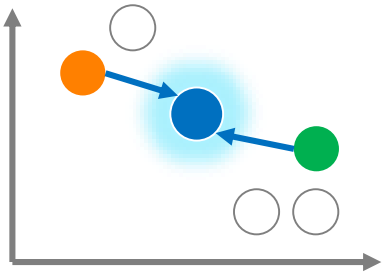
Falco Dürsch
falco.duersch@student.hpi.de
Hasso Plattner Institute
University of Potsdam
Germany

Robert Hirschfeld
hirschfeld@hpi.uni-potsdam.de
Hasso Plattner Institute
University of Potsdam
Germany

ABSTRACT

1 INTRODUCTION

Composition

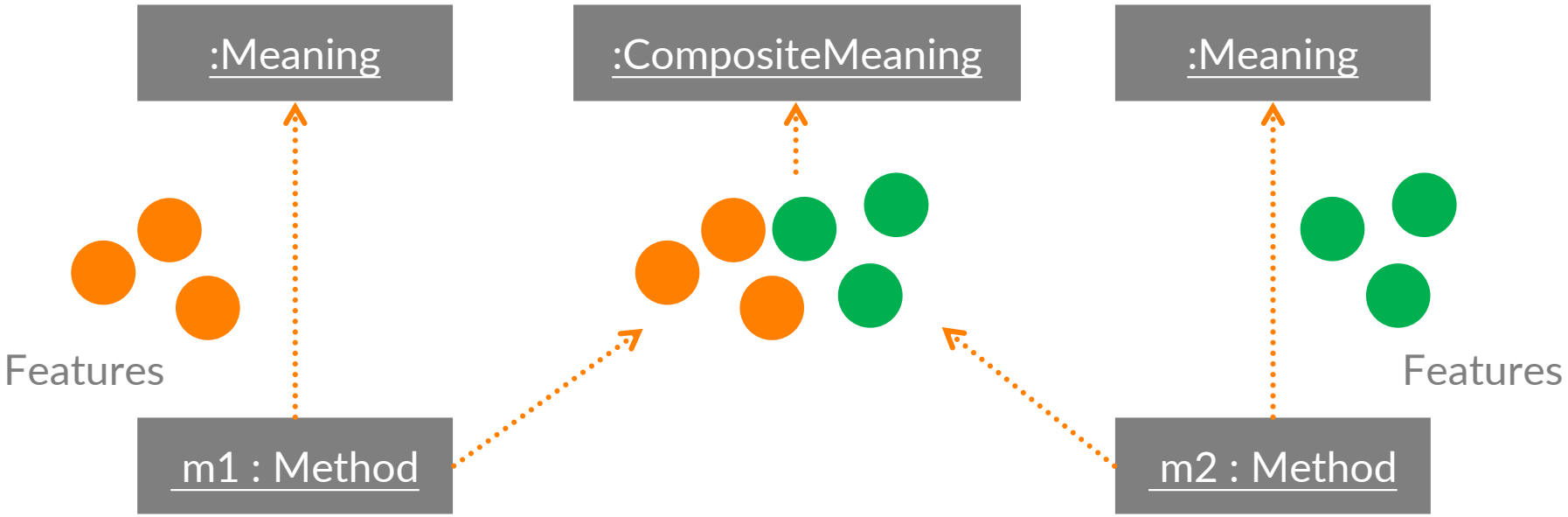


```
m1 meaning composeWith: m2 meaning

Meaning composedFrom: {m1. m2. … }
```

- Represent **classes** as composition of name, fields, doc-string, and methods
- Represent **changes** as composition of modified meta-objects

Destructuring Composition

» Not all models allow **immediate** composition
(e.g. addition in vector space vs. maximum-likelihood)



Use Cases

- Test Prioritization
- Refactoring [s. Paper]
- Metrics/Linting
  - Intra-class similarity ~ cohesion
  - Inter-class similarity ~ coupling
- Code Completion
  - Incomplete code without valid meta-object representation can have a representation in the model
- …

Implementation: Scope

```
model := LDAModel trainOn: #package alpha: 0.05 beta: …
model do: [ “The LDA model is valid here” ]

Model >> do: aBlock
  ^ActiveModel value: self during: aBlock
```

Dynamic variable

```
CompiledMethod >> meaning
  ^ActiveModel value meaningOfMethod: self

Class >> meaning
  ^ActiveModel value meaningOfClass: self
```

Double dispatch back to dynamic variable

```
model meaningOf: anObject
```

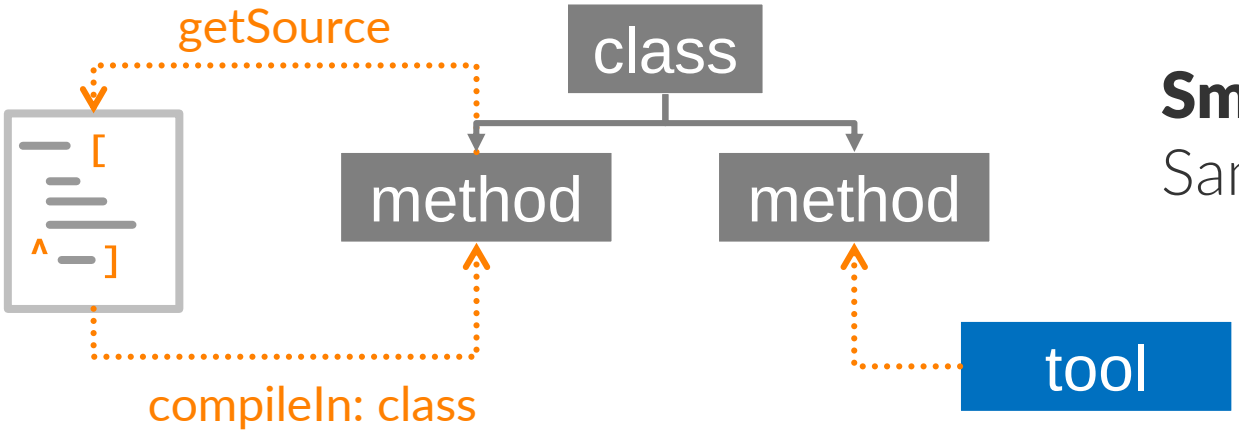
More explicit?

Implementation: Minimal Core

- A Model only needs to…
  - represent features
    (lexical tokens in identifiers, strings, symbols)
  - provide composition and comparison

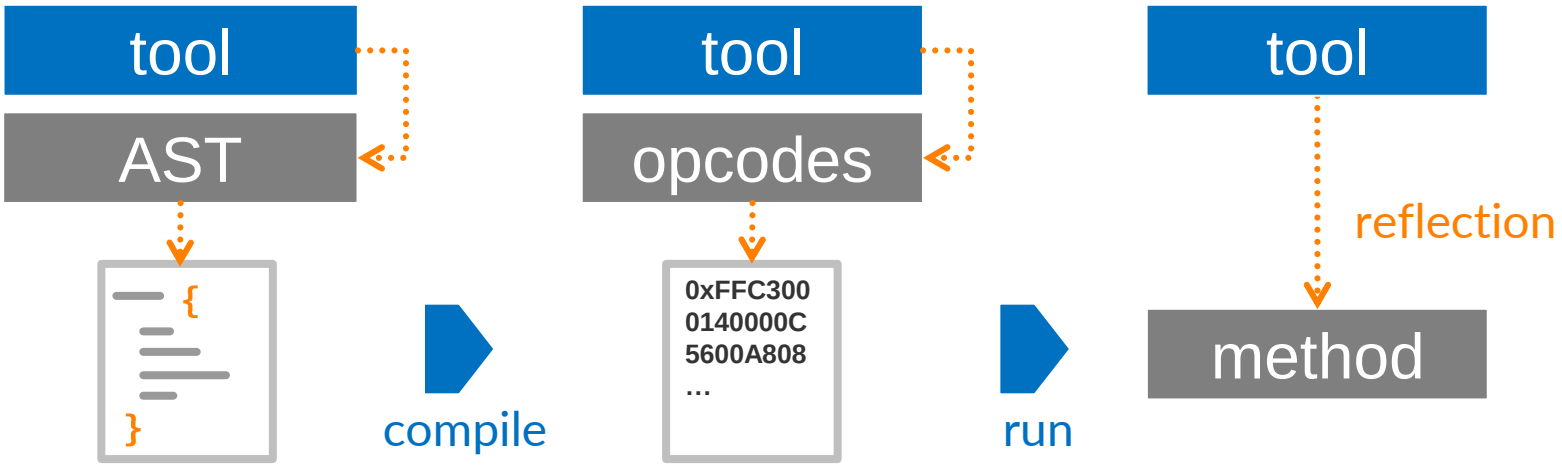
- Default implementations (via double dispatch)
  - Methods as composition of identifiers
  - Classes as composition of name, fields, and methods
  - …

Meta-objects?



Smalltalk
Same life-cycle

Java/.NET/C[++]



Conclusion

- Programs are “natural” artifacts of human communication
- “Natural” properties are usable through ML but not reified in metaprogramming, yet
- We explored designs to expose “ML/NLP knowledge” at meta-object level

- How do we want to use secondary notation & meaning through metaprogramming?